SCI INSTITUTE
TECHNICAL REPORT

A Comparison of Load Balancing Algorithms for AMR in Uintah

Qingyu Meng, Justin Luitjens, Martin Berzins

UUSCI-2008-006

Scientific Computing and Imaging Institute
University of Utah
Salt Lake City, UT 84112 USA

October 27, 2008

Abstract:

This technical report discussed load balancing algorithms for AMR and the implementations of load balancers currently available in Uintah, the multi-physics framework, developed by the Center for Accidental Fires and Explosions (C-SAFE) to aid the simulation of fires and explosions. We designed several experiments to compare these load balancers by the execution time, the balance of work load distribution, the volume of communications and the efficiency. Form the results, we can verify that we implemented a better new space-curve filling load balancing algorithm comparing to Zoltan Load Balancing Library of Sandia National Laboratories. Also through these experiments, it can provide us a detail of how the load balancing effect the overall performance of large scale parallel applications.

THE UNIVERSITY OF UTAH

A Comparison of Load Balancing Algorithms for AMR in Uintah

Qingyu Meng, Justin Luitjens, Martin Berzins

October 16, 2008

1 Introduction

As we have to improve the scalability on large computing system which already comes to petascale with thousands of processors. The load balancing algorithm, controls both the assignment of workload to processors and the interprocessor communications, is obviously critical for parallel computation softwares. We must assign the work load across the processors more balance to lower the processor's waiting time for other processor, and also at the same time, we need to put more computing associated parts together to reduce the communication cost. That means to compute accuracy balancing result is important to improve the performance on large scale machines.

But the cost of load balancing process itself should also be considered, especially now adaptive mesh refinement(AMR) [4] is widely used. When AMR is involved to adaptively increase the resolution of the grid only where it is needed, to reduce the work load, the mesh keeps changing in computation. In this situation we need *dynamic* load balancing(DLB) [7] to assign the workload and it will be called many times according how the mesh is refined. Therefore, a good DLB algorithm should consider both the result accuracy and the partitioning cost of itself.

Our study based on Uintah [5, 10–12], the multi-physics framework, developed by the Center for Accidental Fires and Explosions (C-SAFE) to aid in the simulation of fires and explosions. Uintah uses both parallel and AMR to perform simulations on very large problems with a high degree of accuracy.

2 Dynamic Load Balancing Algorithms

Parallel computing distribute the data among the processors, the owner of that part of data performs all computations on its data and also performs communications to the neighbor processors if data dependencies among data items owned by different processors. So the time of execution contains the time of setting up the data, the time of computing on its own data, the time of communications and the time of waiting others.

The requirement of dynamic load balancing algorithms is to minimize total execution time.

1. Minimizing processor idle time. (the total weight in each partition is approximate equal)

2. Minimizing inter-processor communications. (minimum the total cut-weight of the partition)

3. Minimizing the load balancing time.(the load balancing algorithm should be implemented in parallel and light weight)

4. Minimizing the cost to redistribute data.

There are two major kinds of load balancing algorithms in practice. Geometry algorithm also called mesh based or coordinated algorithm. The computation work on a geometric domain which has may data items, every data items has coordinates to provide their position and weight of computation cost. Geometric locality is loosely correlated to data dependencies.

Graph algorithm work on a domain that no geometry information is needed, but the connectivity among data items and the weight of computation cost of each item is known. This domain can be represented as graph.

2.1 Geometry algorithms

2.1.1 Recursive Bisection Load Balancing

Recursive bisection is a simple and intuitive algorithm. The algorithm choose a direction and then split the graph (mesh) by making an appropriate perpendicular cut. The position of the cut should be such that an equal number of graph vertices (elements) fall on either side of it. The sub-domains are then further divided by recursive application of the same splitting algorithm until the number of partitions equals the number of processors. The most common geometry recursive bisection algorithms are Recursive Coordinate Bisection (RCB) [1] and Recursive Inertial Bisection (RIB) [14]. They use different method to choose the cutting planes.

- RCB: the cutting plane will always be orthogonal to coordinate in turn.
- RIB: the cutting plane will always be orthogonal to the longest direction of the domain, the eigenvector of the inertial matrix.

RCB/RIB uses coordinates, potentially geometrical closed elements are likely to be assigned to the same processor, that can reduce the communications.

2.1.2 Space Filling Curve Load Balancing

Space filling curve(SFC) was purposed to simplify the partition problem by mapping n-dimensional space to one dimension [13]. We need two functions to partition a domain:

1. Curve generate function: map a point (elements) in one, two or three dimensions into a curve.

2. Interval partitioning function: seeks divide the curve into P intervals each containing the same weight of objects associated to these intervals.

Curve generate function will map physically close elements also close on the curve. In this way, the algorithm assigns physically close elements to same processor to reduce the communications.

2.2 Graph algorithms

The computation domain can be represented by an dual communication graph $G(V, E)$. G is an undirected graph, V is a set of vertices represent work, E is a set of edges represent the communication. In this way, the load balancing algorithm can put the exact communication amount into computation instead of assuming the communication by coordinates. So we need also provide communication costs for these algorithms, which makes them more complex.

3 Zotlan Toolkit

Zoltan [6] is a toolkit for parallel load balancing and data management in scientific computing developed by Sandia National Laboratories. It contains bunch of collations of load balancing algorithms, including geometric (coordinate-based) partitioners such as RCB, RIB, HSFC and Refinement Tree. Zoltan also supports Graph, Hypergraph partitioners, some of them implemented by Zoltan itself and others are from some famous third party load balancing library. These features helped Zoltan to build an unified interface, so that the users can easily switch from one load balancing algorithm to another algorithm.

Zotlan provide object-based, call-back function style interface [2] which makes it easier to be integrated in Uintah. For the geometry load balancing algorithms, four call-back functions need to be implemented, which provide the number of elements, the weights array, the number of dimensions and the coordinates array. The desired algorithm can be selected by set param function, and also there are many of additional parameters for each algorithm. After everything is setup, call LB_Partition and this function will perform the partition and return the result.

4 Load Balancers in Uintah

The load balancing procedure in Uintah working with the scheduler determines a reasonable allocation of tasks to processing resources based on predicted weights. These predicted weights are calculated through certain criteria, such as number of particles, number of patches, number of cells, etc. The DLB attempts to guarantee that an equal amount of work is distributed to each processor allowing for optimal scaling of the simulation to multiple processors.

Because we do not predict the commutation costs between the patches, so Uintah only use the geometry partitioners. For now, Uintah provide several load balancing algorithms for user to choice.

These load balancers are implemented in Uintah dynamic load balancing component or by calling the Zoltan library. We will discuss the detail of implementation and compare the following four load balancers.

```
/* Zoltan Initialization*/
Zoltan_Initialize( 0, NULL, &ver );
/* Create Zotlan Object */
zz = new Zoltan( d_myworld->getComm() );
......
/* Choose Zoltan Partition Algorithm */
zz->Set_Param("LB_METHOD", Algorithm);
......
/* Register call-back functions */
zz->Set_Num_Obj_Fn(get_number_of_objects, ...);
zz->Set_Obj_List_Fn(get_object_list, ...);
zz->Set_Num_Geom_Fn(get_number_of_geometry, ...);
zz->Set_Geom_Multi_Fn(get_geometry_list, ...);
......
/* Do partition*/
int rc = zz->LB_Partition(...);
```

Figure 1: Zoltan functions in Uintah

4.1 Zoltan RCB

Zoltan implemented RCB in parallel [3]. Every processor has part of objects in initial, each subdomain of processors and the objects that are contained on those processors are divided into two sets based on which side of the cutting plane each object is on. Either or both of these sets may be empty. After this cut, the left part of all the processors have will be one subdomain for further cut, and right part of all processors will be another subdomain. On every cut, Zoltan RCB will try to search the appropriate cutting plane on every coordinate axes, and choose the coordinate which has the best balancing result. Zoltan SFC is very fast and efficient, but the partition quality is mediocre and may generate disconnected subdomains.

4.2 Zoltan RIB

RIB implementation in Zoltan is very similar to RCB. The processor and associated objects are handled by the same routine as are used by RCB. But RIB need to compute the direction vector for the cutting plane, instead of trying each coordinate directions in RCB. This make Zotlan RIB more complex and the cost of this algorithm is slightly higher, but can product better balancing result compare to RCB.

4.3 Zotlan HSFC

The first phase of this algorithm is to generate the space filling curve. The bounding box is build that contains all of the objects using their two or three dimensional spatial coordinates and also slightly

Name	Implemented by	Description
SingleProcessor	Uintah	Assigns all patches to one processor
Simple	Uintah	Assigns clumps of patches on the order of patches
RoundRobin	Uintah	Assigns patches based on index number mod processor rank
SFC	Uintah	Dynamic Load Balancing algorithm based on space filling curve
RCB	Zoltan	Recursive Coordinate Bisection algorithm
RIB	Zoltan	Recursive Inertial Bisection algorithm
HSFC	Zotlan	Hilbert Space Filling Curve algorithm
Block	Zoltan	Similar to RoundRobin, only use the index number of patches
Random	Zoltan	Randomly assign the pathes, for test and debug

Figure 2: Load balancers in Uintah

expanded to ensure that all objects are strictly interior to the boundary surface. The bounding box is necessary to calculate the table which is used for Hilbert Space Filling curve generation. By looking up the table, curve generate functions then can map a point in one, two or three dimensions into the interval $[0, 1]$ and vice versa.

The second phase of Zoltan HSFC is to divide the curve into P intervals each containing the same weight of objects associated to these intervals. The unit interval is divided into $k(P-1)+1$ bins, where k is a small positive constant, P is the number of processors. These bins are set to equal size in initial and form a non-overlapping cover of $[0, 1]$. So each bin has an left closed endpoint and right open endpoint $[l, r)$, this half-open interval represent the bin. The flowing of this algorithm is essentially to continue to refine the boundary of each intervals, to make every part has some weight as possible.

On every step of integrations, an MPI_Allreduce call is made to globally sum the weights in each bin and also found the maximum and minimum coordinate in each bin. Zoltan HSFC then calculate the average weight of P intervals then use greedy algorithm to sum the weights of the bins from left to right until the next bin would cause the weight of this part is more than this average weight. This bin in the boundary is called "overflowing" bin. The location of each cut before an "overflowing" bin, and the size of its "overflowing" bin are saved. The algorithm try to divide the overflow bins to balance the weight. If the bin's maximum and minimum coordinates are too close relative to double precision resolution, the bin can not be practically subdivided. The iteration will continue, until no bin can be further divided. In this progress, the bounder of each partition become more and more clear and keep weight of every intervals approximate equal.

After these two phases, by looking up the table, these intervals can map back to the original coordinates.

4.4 Unitah SFC

The space filling curve generating phase of Uintah SFC is basically the same with Zotlan HSFC, they both use HSFC function to generate the curve. Uintah SFC does this through a highly parallel method [9].

	Uintah SFC	Zoltan HSFC
Space Filing Curve	Hilbert Curve	Hilbert Curve
Curve generate	Divided & Sorting	Table Driven
Partition method	Parallel	Parallel
Object size	Constant	Combine & Split
Iterate operation	Reduce MaxCost and Imbalance locally	Refine the interval boundary
Iterate halt condition	MaxCost not reduce	Bin cannot be further divided

Figure 3: Comparison of Uintah SFC vs Zoltan HSFC

Each processor is given a subset of the patches to sort according to the space filling curve. While sorting the subset of patches each processor generates a unique curve index for each of its patches that are then used to merge the curves from each processor into a single curve instead of a table driven logic used by Zoltan HSFC.

In the curve distribution phase, Uintah SFC linearly traverses the patches along the curve while assigning them to processors. There are two variables to guide the partition of the curve: imbalance value and max cost threshold. The imbalance value can be calculated by the weight of work assigned to the current processor and the average weight of unassigned work for the remaining processors. When Unitah SFC work through the curve, it continues to assign the work to current processors until next point will cause the imbalance value to raise. Then this point will be assigned to next processors and begin the assignment process of that processor.

The other condition in the curve partition is to attempt to minimize the maximum assigned work by not allowing the work on any processor to exceed maxcost threshold W_{max}. This threshold is updated every iteration, we can control W_{max} will not increase on every iteration, so that every step will make progress. When max cost cannot reduce, the iteration will be stopped, the phase of partition the curve is completed.

5 Testing experiment

The load balancing experiments ran on Ranger of TACC. Ranger has four AMD Opteron Quad-Core 64-bit processors on each node. The interconnection of the system is Infiniband, and Ranger also use Lustre filesystem run over the Infiniband. We tested four load balancing algorithms, Zoltan RCB, Zoltan RIB, Zoltan HSFC and Uintah SFC by solving two ICE [8] problems.

5.1 Measurement

To measure the load balancer, we need to determine the quality of load balancing result, the cost of load balancing algorithm and most important the performance and scalability of whole application. The quality of load balancing result can be described in two sides: how equal of the weight in each partition and how much communications across these partitions. In this article, we use $imbalance$ to measure the work load distribution, and use both $messages$ and $datavolume$ to measure the communications.

Imbalance

The imbalance is defined as:

$$I = \frac{W_{max}}{W_{avg}} - 1$$

Where W_{max} is the maximum weight of result partitions and W_{avg} is the average weight of the result partitions. In ideal scene, when every partition has equal weight, $W_{max} = W_{avg}$, I will equal to zero. W_{avg} is determined by the total weights of the whole domain, while load balancer intended minimum the W_{max} to reduce the idle time wasted for waiting other partitions. So the less imbalance value indicates the weight distribution result is better. We collected the weights of all partitions once load balancer was called, calculated the imbalance value and stored it to output file.

Number of messages and data volume

Unlike imbalance is calculated after every load balancing, in our experiment communication messages and volume are collected during the execution of each partition. We use two counters on every rank to record the MPI message sends, one for the number of messages and another one for the data volume of that communication.

$$\text{Data volume on rank}(i) : D_i = \sum_{k=1}^{N_i} S_k, \text{ Number of messages on rank } (i) : N_i$$

Where S_k is the message size of that particular MPI call.

At the end of every simulation time step, MPI_Allreduce will be called to calculate the average and maximum number of messages (N_{avg} and N_{max}), and average and maximum data volume (D_{avg} and D_{max}). The real cost of communication is related both the number of messages and data volumes:

$$T_i = \sum_{k=1}^{N_i} T_{startup} + S_k/Bandwith = N_i * T_{startup} + V_i/Bandwith$$

$T_{startup}$ and $Bandwith$ are determined by the architecture and network topology, but it will be a good load balancing result if we can lower both the number of messages and the data volumes.

Component timing

Component timing can help us to find out the cost of load balancer itself, the execution time of task , the overhead of data preparation and the time wasted on waiting for data. We insert serval timers in the source code to generate the detail time cost for each component. Also MPI_Barrier is inserted when needed to make the measurement more accuracy.

5.2 Some Issues

Serval issues was solved in the experiment phase. First, MVAPICH 1.0 on Ranger do not correctly release memory associated with the communicator during MPI_Comm_free. This will lead to a memory leek resulting in growing memory use over multiple invocations of RCB or RIB in Zoltan library. Because Zoltan partitioning methods RCB and RIB use MPI_Comm_dup and MPI_Comm_split to recursively create communicators with subsets of processors. Fortunately, there is an undocumented parameter in Zoltan, which can use an alternate implementation that doesn't use MPI_Comm_split to be invoked.

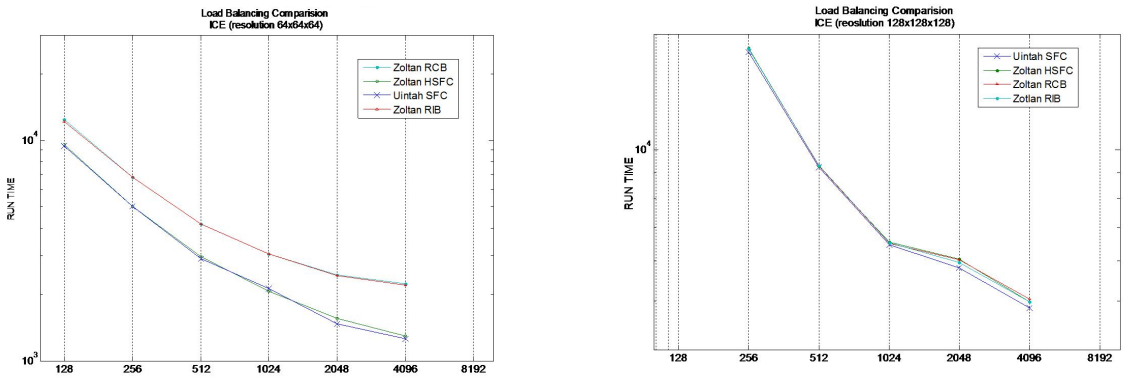

Figure 4: Total Runtime of two ICE problem, resolutions: 64^3(left) 128^3(right)

Second, a potential deadlock is founded in Zoltan library. This bug can been seen in Ranger machine when running with more than 1024 processors. A paired MPI_Send and MPI_Recv is not well ordered when creating processor lists in RCB or RIB algorithm. This bug have been submitted to Zoltan development team.

6 Results

From the result generated at the testing phase, we will look through how well the requirements discussed on the second part are accomplished by the four load balancing algorithms and how these factors effect the overall performance and scalability of Uintah.

6.1 Overall runtime and scalability

Figure. 4 shows the scalability comparison of these four algorithms, these scaling data are the overall runtime of Uintah program on processors from 128 to 4096 on two different resolutions of ICE problem. This figure is a loglog based graph which can represent the scaling trend well but lost the detail of comparison . Figure. 5 is produced to show the runtime comparison of these algorithms by normalizing based on the execution time of Uintah SFC.

From the graph, we can see for the overall performance on execution time, Uintah SFC is significant better than Zoltan HSFC. But in some case, Zotlan RCB and Zoltan RIB drive better performance.

6.2 Workload distribution and Imbalance result

The result of workload distribution can help us to understand where the significant difference of Uintah SFC and Zoltan HSFC comes from. Figure. 4 shows the weight assigned to each rank after load bal-

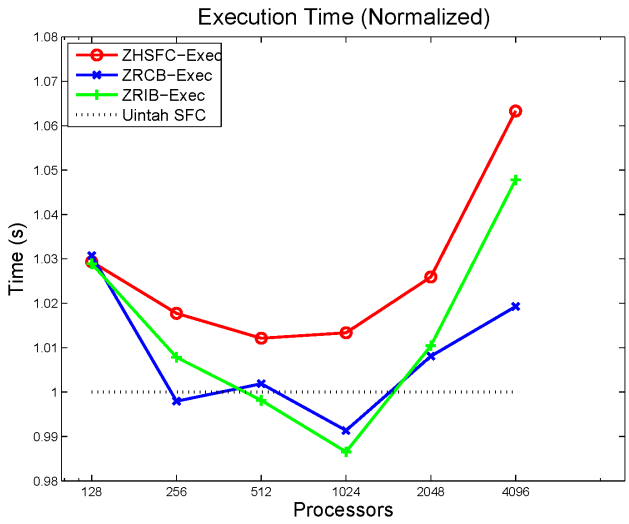

Figure 5: Execution time comparison, normalized based on Uintah SFC

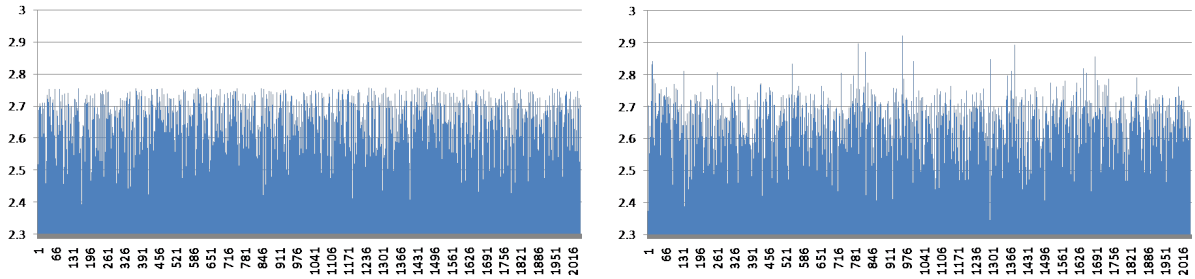

Figure 6: The work load distribution over 2048 processors, Uintah SFC(left) and Zoltan HSFC(right)

ancing, the data is generated on a 2048 processor run. X axis represents the number of rank and Y axis represents the weight of assignment.

We can clearly see the figure of Uintah SFC is more smooth than Zoltan HSFC. All the weights of Uintah SFC is limited by the threshold according to the algorithm, but Zoltan HSFC does not have such threshold and produced some very high weight on particular rank. As our goal is to limited the max weight, Uintah SFC does a better job. To make a quantify comparison, Figure. 7 shows the imbalance value runs from 64 to 4096 processors. We can clearly see the imbalance value of Uintah SFC is smaller than Zoltan SFC, which means the result of Uintah SFC is more balanced.

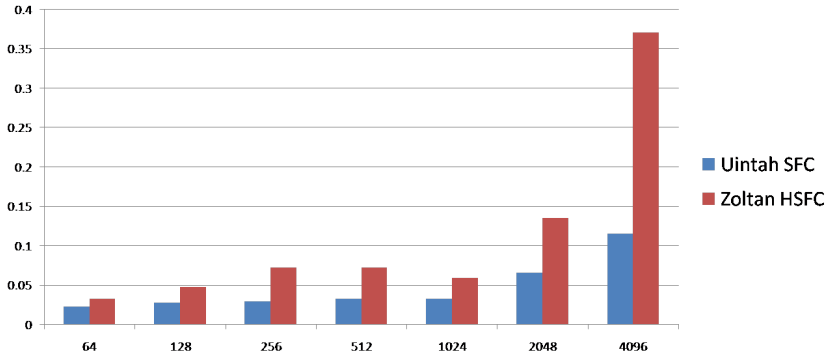

Figure 7: The imbalance value from 64 to 4096 processors

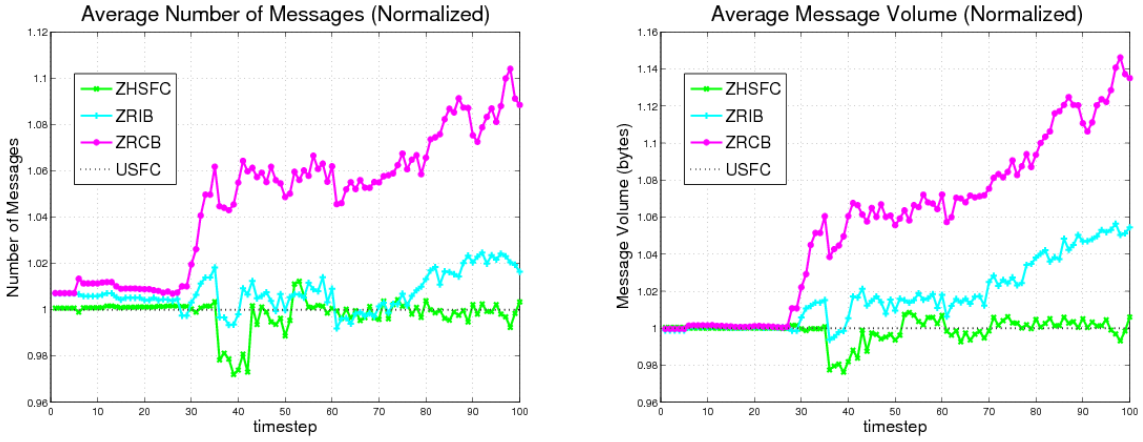

Figure 8: The number of messages and data volumes during the simulation

6.3 Communications

According to the Figure. 8, from both number of messages and data volumes view, the two space-filling curve algorithms are the best and track each other remarkably well. Zoltan RCB is the worst and Zoltan RIB does fairly well but appears to be getting worse toward the end of the run. Clearly, SFC load balancing algorithms will give a better communication result comparing to RB based algorithm.

6.4 Load balancing algorithm cost

Figure. 9 shows the execution time of load balancer itself from 128 to 4096 processors. For the two space-filling curve algorithms, Uintah SFC algorithm is clearly better than Zoltan HSFC algorithm. Zoltan's HSFC algorithm cost more than four times than Uintah SFC on 4096 processors. For the two RB based algorithms, as we expected, these two algorithms are every light weight and efficient.

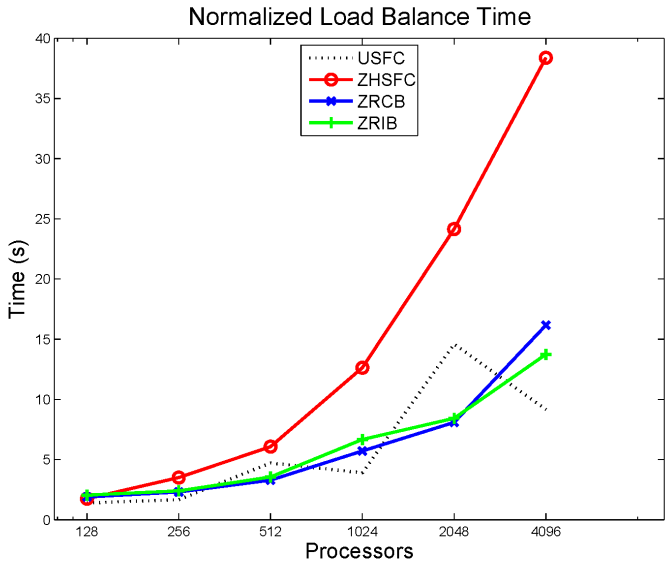

Figure 9: Comparison of load balancer running time.

7 Summary

Uintah implemented a new SFC load balance algorithm through sorting and heuristic threshold. We integrated Zoltan load balancers into Uintah and compared with our SFC algorithm. Through the comparison, we explained and observed different character and behavior of these algorithms on aspects of work load assignment, communications, algorithm cost and effect to performance and scalability of Uintah. From the result of these comparison, we verified that Uintah SFC algorithm can produce better load balancing partitions than Zoltan HSFC, also the cost of Uintah SFC is much lower.

8 Acknowledgement

This work was supported by the University of Utah's Center for the Simulation of Accidental Fires and Explosions (C-SAFE) and funded by both the Department of Energy, under subcontract No. B524196 and the National Science Foundation under subcontract No. OCI0721659.

References

[1] M. J. Berger and S. H. Bokhari. A partitioning strategy for nonuniform problems on multiprocessors. *IEEE Trans. Comput.*, 36(5):570–580, 1987.

[2] Erik Boman, Karen Devine, Robert Heaphy, Bruce Hendrickson, Vitus Leung, Lee Ann Riesen, Courtenay Vaughan, Umit Catalyurek, Doruk Bozdag, William Mitchell, and James Teresco. *Zoltan 3.0: Parallel Partitioning, Load Balancing, and Data-Management Services; User's Guide*. Sandia National Laboratories, Albuquerque, NM, 2007. Tech. Report SAND2007-4748W `http://www.cs.sandia.gov/Zoltan/ug_html/ug.html`.

[3] Erik Boman, Karen Devine, Robert Heaphy, Bruce Hendrickson, Vitus Leung, Lee Ann Riesen, Courtenay Vaughan, Umit Catalyurek, Doruk Bozdag, William Mitchell, and James Teresco. *Zoltan 3.0: Parallel Partitioning, Load Balancing, and Data-Management Services; Developer's Guide*. Sandia National Laboratories, Albuquerque, NM, 2007. Tech. Report SAND2007-4749W `http://www.cs.sandia.gov/Zoltan/dev_html/dev.html`.

[4] Phillip Colella, John Bell, Noel Keen, Terry Ligocki, Michael Lijewski, and Brian van Straalen. Performance and scaling of locally-structured grid methods for partial differential equations. *Journal of Physics: Conference Series*, 78:012013 (13pp), 2007.

[5] J. Davison, St. Germain, John Mccorquodale, Steven G. Parker, and Christopher R. Johnson. Uintah: A massively parallel problem solving environment, 2000.

[6] Karen Devine, Erik Boman, Robert Heaphy, Bruce Hendrickson, and Courtenay Vaughan. Zoltan data management services for parallel dynamic applications. *Computing in Science and Engineering*, 4(2):90–97, 2002.

[7] Karen D. Devine, Erik G. Boman, Robert T. Heaphy, Bruce A. Hendrickson, James D. Teresco, Jamal Faik, Joseph E. Flaherty, and Luis G. Gervasio. New challenges in dynamic load balancing. *Appl. Numer. Math*, 52:2005, 2004.

[8] Thomas C. Henderson, Patrick A. McMurtry, Philip J. Smith, Gregory A. Voth, Charles A. Wight, and David W. Pershing. Simulating accidental fires and explosions. *Comput. Sci. Eng.*, 2(2):64–76, 2000.

[9] J. Luitjens, M. Berzins, and T. Henderson. Parallel space-filling curve generation through sorting: Research articles. *Concurr. Comput. : Pract. Exper.*, 19(10):1387–1402, 2007.

[10] J. Luitjens, B. Worthen, M. Berzins, and T.C. Henderson. Scalable parallel amr for the uintah multiphysics code. In D. Bader, editor, *Petascale Computing Algorithms and Applications*. Chapman and Hall/CRC, 2007.

[11] Steven G. Parker. A component-based architecture for parallel multi-physics pde simulation. *Future Gener. Comput. Syst.*, 22(1):204–216, 2006.

[12] Steven G. Parker, James Guilkey, and Todd Harman. A component-based parallel infrastructure for the simulation of fluid structure interaction. *Eng. with Comput.*, 22(3):277–292, 2006.

[13] Hans Sagan. *Space-Filling Curves*. Springer-Verlag, 1994. ISBN 0387942653.

[14] Valerie E. Taylor and Bahram Nour-omid. A study of the factorization fill-in for a parallel implementation of the finite element method. *Int. J. Numer. Meth. Engng*, 37:3809–3823, 1994.